APPENDIX B

TECHNICAL WHITE PAPER: NETCENTRIC VIRTUAL SUPERCOMPUTING

VERSION 1.21 REV DECEMBER 13 2001

(28 pages)

Technical White Paper

NetCentric Virtual Supercomputing*†

A novel software solution for transforming a standard commercial network of conventional computers into a general-purpose virtual machine capable of multi-gigaflop computational throughput.

Dr. Charles C. Wurtz & Dr. Gary C. Berkowitz

Veriscape, Inc
The Netcentric Edge ™

26 April 1999

12 May 1999 (v 1.10)
31 Jan 2000 (v 1.20)
05 Mar 2000 (v 1.21)

* The authors wish to thank Dr. Gerald Ferrentino of *SalomonSmithBarney*, Dr. John Blin of *Advanced Portfolio Technology*, and Dr. Camilo Gomez of *The Center for Adaptive Systems Applications* & *Los Alamos National Laboratory* for extremely valuable comments and consultation.

Outline of Contents

Target Audience

With the exception of the Executive Overview, this White Paper is intended for readers with a graduate level (or equivalent) background in mathematics and computer science. For a complete appreciation of the *RiskScape*™ Intelligent Financial Risk Management Application, some background in quantitative finance is required. An understanding of current risk control methodologies such as Value-at-Risk and Portfolio Stress-Testing (as well as their limitations) is necessary.

Abstract of the White Paper

This White Paper advances and details the concept of implementing a novel software architecture on a heterogeneous network of conventional computational platforms to create a Netcentric Virtual Supercomputer Infrastructure (NVSI). In order to provide a full appreciation for the commercial importance of this breakthrough enabling technology, the Paper also describes one possible application of the NVSI, named *RiskScape*. The *RiskScape* application addresses a particular computationally-intensive problem in financial risk management known as *Portfolio Stress-Testing*.

We begin by discussing why this technology represents a breakthrough, and we then enumerate the several innovations that we have brought to bear in order to achieve this breakthrough. We next discuss the nature of how NVSI accomplishes its radical level of performance without special-purpose hardware. We then review the reasons why we believe no one has heretofore taken this approach to computer system design.

The paper continues with a mathematical treatment of the *RiskScape* design in order to demonstrate the power of the NVSI technology. In this section we outline the assumptions we have made to make the problem computationally tractable, as well as discuss several of the optimization techniques that the NVSI platform provides to the application developer.

Finally we make a comparison of the NVSI technology with hardware-based supercomputers. This comparison addresses both the classes of problems to which NVSI is suited, as well as a discussion of those to which it is not. The last comparison is in price/performance. Here we show the estimated computational throughput of the NVSI system in terms of VFLOPS (Virtual Floating-point Operations per Second), and compare this with two *Cray Research*® machines, the *T90* and the *T3E*.

Our findings conclude that for certain classes of problems for which pre-computation is a viable methodology, the NVSI/*RiskScape* solution is capable of about 250 (Virtual) GigaFLOPS. This performance is comparable to that of the *Cray T3E*. Most importantly from a commercial perspective, we estimate that the cost of an NVSI/*RiskScape* implementation will be on the order of 50 to 100 times less than a comparable hardware-based solution.

Executive Overview

Netcentric computing is the new paradigm for building efficient, cost effective computer systems to solve numerous business problems. Many commercial enterprises have invested substantial sums in computer hardware only to discover that they realize a fraction of the total CPU power. This is because the operating costs of a piece of hardware are identical whether the machine is running at its peak capacity, or sitting idle. The problem is compounded by the fact that while one machine is sitting idle, another in the same office is so overloaded that it has slowed to a crawl. What is needed is a way to allow the power of idle, or underutilized, machines to automatically augment the capacity of those that are over burdened. Such a solution would allow businesses to add hardware in an incremental fashion, rather than having to continuously upgrade expensive servers and mainframes.

This White Paper describes a breakthrough, software-based enabling technology that transforms a network of conventional PCs, workstations and servers into a virtual supercomputer designed for optimal performance over a wide class of commercial domains. Quite general in nature, this virtual supercomputer can be used to solve many (although certainly not all) computationally intensive problems that are found in a number of different businesses. These include telecommunications switching, investment-portfolio valuation, e*commerce order processing, high-demand query caching, and fraud detection, to name a few.

So that our proposed technology does not begin life as a solution in search of a problem, we have applied virtual supercomputing to the problem of Financial Risk Management. Specifically, we have addressed *Portfolio Stress-Testing* an area that has a significant need for a cost-effective solution, and for which our new technology is well suited.

However, it must be appreciated from the outset that while we describe a specific application, this should in no way imply that the underlying technology is limited to this application. Our sole purpose in blending the description of the risk management application with the enabling technology in this paper is to demonstrate the vast cost/performance benefit.

RiskScape: An Application of Netcentric Supercomputing to a Business Problem

It is clear that many commercial problems, such as Financial Risk Management, need the level of computational power associated with conventional hardware-based supercomputers. Unfortunately, unlike mission-critical projects for military and government operations, the commercial world is highly constrained by economic considerations. In the main, businesses cannot justify the expenditure of tens, if not hundreds, of millions of dollars on computer hardware that will become obsolete in three to five years. This is true even though rapid and accurate risk management can spell the difference between business success and catastrophic failure.

What is needed for commerce is a viable solution that transforms the substantial hardware investment already made by a firm into a computational platform capable of supporting the necessary time-critical decision process. In this paper we outline our design for such a computer system (termed *Netcentric Virtual Supercomputer Infrastructure* or NVSI), and then continue to describe one commercial application (*RiskScape*™) that together can provide the level of performance financial institutions require in order to manage their risk.

Our current best estimate of the sustained "Virtual FLoating-point OPerations per Second" (VFLOPS) obtainable by our risk-management application supported by our proposed NVSI technology, is between 5 and 250 *GigaFLOP* (billion floating-point operations per second), depending on the number of CPUs available to the system. This meets the computational power of a *Cray Research T3E*, the highest-end commercially available supercomputer.

This translates as the ability to evaluate a portfolio consisting of **one million instruments** (including portfolios of derivatives on both debt and equity instruments) **across three million scenarios** in **under one hour.**

It is important to note that the solution we propose is software based, and requires only an incremental amount of additional hardware. From an economic and commercial standpoint we estimate that a full *NVSI/RiskScape* implementation will be between 5 and 10 million dollars. A hardware-based solution with comparable performance would cost in excess of 50 to 100 million. Moreover, unlike a hardware solution, the NVSI system **will not become obsolete with advances in computer technology**. Indeed, performance only improves with evolution in platform and network capability.

RiskScape™

Dragon Hunting in the Real World

Using NVSI for on-demand stress-evaluation of portfolio risk via large-scale projection of instrument trajectories over a systematic spectrum of path-dependent scenarios, on a landscape of unexpected events.

I. Background

Computing portfolio risk (PR) is a problem taking center stage in investment risk management. Managing risk is not trivial. Banks and other financial institutions are in the business of taking risks to generate increased revenues and profits, but they are also required to protect shareholder value and prevent catastrophic losses from occurring. To date, evaluation of PR has mostly been confined to macroscopic aggregate measures such as Value-at-Risk (VaR), which estimates PR as an expected loss derived from a weighted sum of volatilities in the individual securities in the portfolio, based on small, statistically-derived market moves. Current risk management systems thus do an effective job of characterizing the expected loss in linear portfolios operating in normal markets. Indeed, some form of full Monte Carlo VaR is the state-of-the-art for both market and credit risk measurement [Jorion 97, Lawrence 96].

Yet, VaR indicates only the maximum expected loss that could occur over some time interval (the portfolio holding period) within some confidence level (usually 2σ or standard deviations, about 97.5%). VaR has nothing to say about discontinuous or extreme (3σ+) market events, such as the Russian Sovereign Debt, and Japanese and Emerging Market currency crises. That is, VaR ignores the "fat tails" in the distribution of portfolio values within which lurk the "dragons" of risk, the unexpected large moves in financial variables that can cause substantial losses, as have been suffered by a number of leading financial institutions over the last several years (such as LTCM). And given that the financial markets are not normally distributed (they are log-normal), such events happen with considerable – and distressing – regularity. To cover the possibility of extreme events, financial institutions implement large safety factors, as the absence of specific risk data for the "tails" induces great caution. The result is that excess capital is lying dormant, an inefficient solution at best. Moreover, if the VaR method is pushed to achieve a greater range of application, severe computational limitations arise. Finally, even setting this consideration aside, VaR calculations still do not really address the "What if?" scenario questions.

In light of these limitations, the method of *stress-testing* has evolved as a complement to VaR. Also known as scenario analysis, this approach attempts to address the weaknesses in VaR by subjectively generating scenarios that simulate large-variance events. This enables the handling of nonlinear positions, and certainly fills in some of the gaps. But as Jorion points out, current implementations of stress-testing are flawed because of the small number of scenarios that can be examined (due to computational constraints), thus forcing necessarily subjective choices about which extreme changes to evaluate. The method also considers movements in only one, or few variables, and correlations are virtually ignored. And most glaring of all the defects in current stress-analysis of PR is the inability to

forecast path-dependent scenarios several time-steps into the future, again due to limits on computational resources.

Even if the appropriate predictive tools existed (such as comprehensive scenario-analysis systems that accessed rich historical data combined with "mark-to-future" states), the sheer data problem is enormous. Further, the integrity of the data is crucial for assuring the validity and integrity of the risk management process results. Real issues of analytical model fidelity and accuracy compound these challenges. New financial products are finding their way to market (credit derivatives, synthetic financial products, etc.) at an accelerating pace. As the worlds of market risk and credit risk begin to merge, there is an accelerating pace in transaction volume growth. It is increasingly clear that the standard approaches to managing risk are not keeping pace with the problem domain. The approach to problem solving in the risk management marketplace today is gated by problems of computing power; database transaction processing throughput, application design, application scalability, and user interface technology [Berkowitz & Wurtz 98].

Of course, this is not news. It is well known that the ideal approach would be to simulate the detailed price trajectory for all the instruments in a portfolio, over a broad range of path-dependent scenarios, using the best (perhaps several) pricing models available (or Monte Carlo simulation, finite-difference methods, numerical integration, or tree expansion, where closed-form models do not exist), all calibrated by accumulations of historical data to provide correlation coefficients, scaling factors, and transition-probabilities for variations in financial parameters. The difficulty has been that such an approach requires an enormous amount of computing power (on the order of leading-edge supercomputers), at a cost that is daunting to even the most resource-rich investment banks. So, we settle for VaR, and a lot of theoretical modeling and projection. But, as pointed out by Wurtz [98], risk analysis must be essentially data-driven, not a theory-driven exercise performed in a data vacuum.

This paper outlines a new technology that allows, for the first time, a practical solution to the problem of calculating future risk for large-scale portfolios. We present a novel computing architecture – essentially a software-based "virtual supercomputer" – that supports on-demand access to projected prices over portfolios of $O(1M)$ securities, for a full range of path-dependent scenarios that entail large ($3\sigma+$) moves in financial variables. Termed *NVSI* (Netcentric Virtual Supercomputer Infrastructure), the system is a suite of highly-optimized system kernels and user applications, designed to emulate the key aspects of supercomputer architecture (tools, techniques, and algorithms), running on off-the-shelf workstation and network hardware, for about $1/100^{th}$ the overall cost of a dedicated supercomputing system.

How does it work?

In brief, by continuous off-line (background) computing, the *RiskScape*/NVSI constructs a daily updated landscape of projected portfolio values for a broad range of scenarios, along with associated scenario probabilities. This multidimensional state-space (hyperspace) can then be queried to yield near-real-time answers to questions such as: which scenarios (if any) could result in catastrophic loss to my portfolio in a week, ninety days, and six months out, and with what likelihood?

By using supercomputing techniques emulated in software, model optimization, massive non-swappable RAM, and distributed processing over commercial networks of existing workstations, the NVSI first populates a hyperspace of up to 10 billion nodes with state-vectors that contain the pricing information (and other moments) for the entire range of instruments in a portfolio of up to 10 million securities.

Once the hyperspace is fully populated with pricing vectors, the portfolio can then be marked to any future state (scenario) desired, simply by looking up (or navigating to) the address of that state and applying the pricing vectors to each instrument. The computational overhead of this second phase is minimal, as the vectors have been pre-computed off-line. Clearly, the combinatoric nature of the problem requires that the solution space be properly constrained. To do this correctly, the optimal granularity and distribution of the state-vectors in the hyperspace must be determined to ensure that the problem domain is fully bracketed.

If the set of state-vectors is chosen appropriately, the entire landscape of scenarios can be searched for the extreme events. In fact. the application can be programmed to find the boundary conditions for catastrophic loss. In other words, the NVSI system can actively search future state-space to determine what combination of market and/or credit conditions would cause the portfolio (or institution) to fail. In addition, the probability of these conditions obtaining, and the likely amount of time required for the conditions to obtain as well as the transition states through which the world must pass to reach the final state, can be determined.

Such information would allow strategic management to see problematic conditions in advance, and take appropriate action. Problematic financial and non-financial holdings can be analyzed, understood and unwound before potential financial meltdowns occur.

Why hasn't this been done before?

The answer is, because the computing demand is enormous, the price/performance ratio for available technology has just recently come within practical range, and no one has heretofore applied a blend of software solutions using virtual-supercomputer architecture, off-the-shelf network hardware, background computing, a spectrum of numerical optimization techniques, and domain-specific "tricks" to make the computational problems more tractable.

Large-scale computing problems have been around since the Manhattan Project, and indeed, Los Alamos was the necessity that mothered the invention of DBM (colloquially named "Dah Big Machine"). Until recently, development of DBMs, now known as supercomputers, has been driven by military and civilian government needs, and hence, contracts. Typical application domains have been weather forecasting, code breaking, signal and image processing, intelligence evaluation, aerospace engineering, and nuclear weapons design. The resulting machines were designed and built with cost as no object, and their prices reflect that history. Spending $50M-$150M on a dedicated number-cruncher is an expense difficult to justify for an investment bank, especially when that glistening rocket ship will become a burdensome dinosaur in about three years.

The alternative has been to scale-up existing software that calculates trajectories for individual instruments, such as options and other derivatives, using pricing models like Black-Scholes. These programs work perfectly well, and provide a flexible platform for future improvements, except that they provide single-instrument answers for one input vector in time-frames on the order of a minute, or at best, several seconds. This is fine for the trader negotiating a position, but when this performance is expanded to include a family of trajectories arising from a wide range of scenarios, and then further multiplied by up to a million instruments in a large global portfolio, even brave souls pale. Waiting a hundred days or more to get an answer somewhat moots the requirement for daily update and response.

The application of the NVSI techniques, in coordination with the recent convergence of several other factors, enables the construction of an affordable virtual-supercomputing system that can usefully meet the need for large-scale, near-real-time portfolio risk analysis. These recent factors are:

- The increased processing speed of affordable workstations,
- Improvements in net-centric computing software,
- The availability of 1GB DRAM chips, thus allowing 128GB (40-bit address) or more of RAM in a single box, and
- Increased demand for a solution due to the increased rate of large bank failures (attributable to a lack of sufficient stress-testing).

In essence, no one has demonstrated a software-emulated supercomputer, because the performance would be too slow for the real-time government applications listed above, and providing a one-off solution for near-real-time commercial needs would require massive construction of custom software at a cost nearly as burdensome as buying a big machine.

Instead, we have determined that it is possible to build a suite of leading-edge system programs and software applications, running on networks of off-the-shelf workstations, that gives dedicated, near-supercomputer performance without requiring much (or eventually, any) specialized hardware. Such an approach can evolve with improvements in hardware and software technology. When presented as a retail solution to potential client financial institutions at a reasonable cost ($1/100^{th}$ that of a dedicated supercomputer), the NVSI system becomes a viable product.

II. What is new here?

What is *not* new is netcentric computing of large-scale problems.[1] The innovation is in building integrated, highly-optimized software that emulates the kit of hardware supercomputing tools and techniques, to create a hardware-independent "virtual supercomputer", optimized to solve a wide class of problems that require large-scale evaluation of independent state-functions in an unbounded hyperspace of multidimensional inputs and outputs. Indeed, our architecture could be used to solve a range of similar problems in other domains, such as credit risk, query-caching, and transaction-processing for global compliance management on the internet, which are decomposable into separate processes of background pre-computation and real-time (demand-query) navigation of a large state-space. NVSI would not be suitable for a domain that required the system to "keep up" with a data stream from the "world" arriving in real-time, such as cryptographic analysis.

The following innovations and/or breakthroughs in *RiskScape*/NVSI are detailed in subsequent sections:

Innovations in Computational design:

- Flexible-structure, fully-compacted, variable-length data words, optimizable for specifiable problem domains;
- Flexible connectivity to allow optimal hyperspatial topology (graphs, trees, hypercubes) relative to a spectrum of specifiable problem domains;
- Highly-optimized numerical techniques for moderate-accuracy computation;

[1] This has also been termed "Metacomputing" by the NCSA (National Center for Supercomputing Applications) and its affiliated institutions, and is being actively pursued as an alternative to single-box supercomputing, using wide-area networks of high-end mainframes.

- Software emulation of supercomputing structures and processes (such as simple, efficient data representation and handling; inherent vector representation, limited data/computation modes, interleaved memory, table lookup, induced pointers, and distributed & parallelized computation [Aho *et al* 74, Hillis 85]), thus providing cost-effective scaling and enhancement;
- Separation of processes into pre-computation (populating the state-space) and navigation (searching the resulting hyperspace of results);
- Second-order "dæmon" processing to interpolate with finer granularity (mesh enhancement) around selected nodes in state-space.

Innovations in Computational Risk Analysis:

- Simulating only extreme (3σ+) moves in financial variables;
- Optimized model representations and pre-computed parametric function spaces;
- Using virtual or *proxy* instruments to represent whole classes of securities – such as cash, options and currency swaps – that share the same basis (underlying asset or index);
- Statistical sampling to create a much smaller representative portfolio.

III. Overview of the NVSI Architecture

The functional block-diagram of the NVSI illustrates the essential aspects of the design:

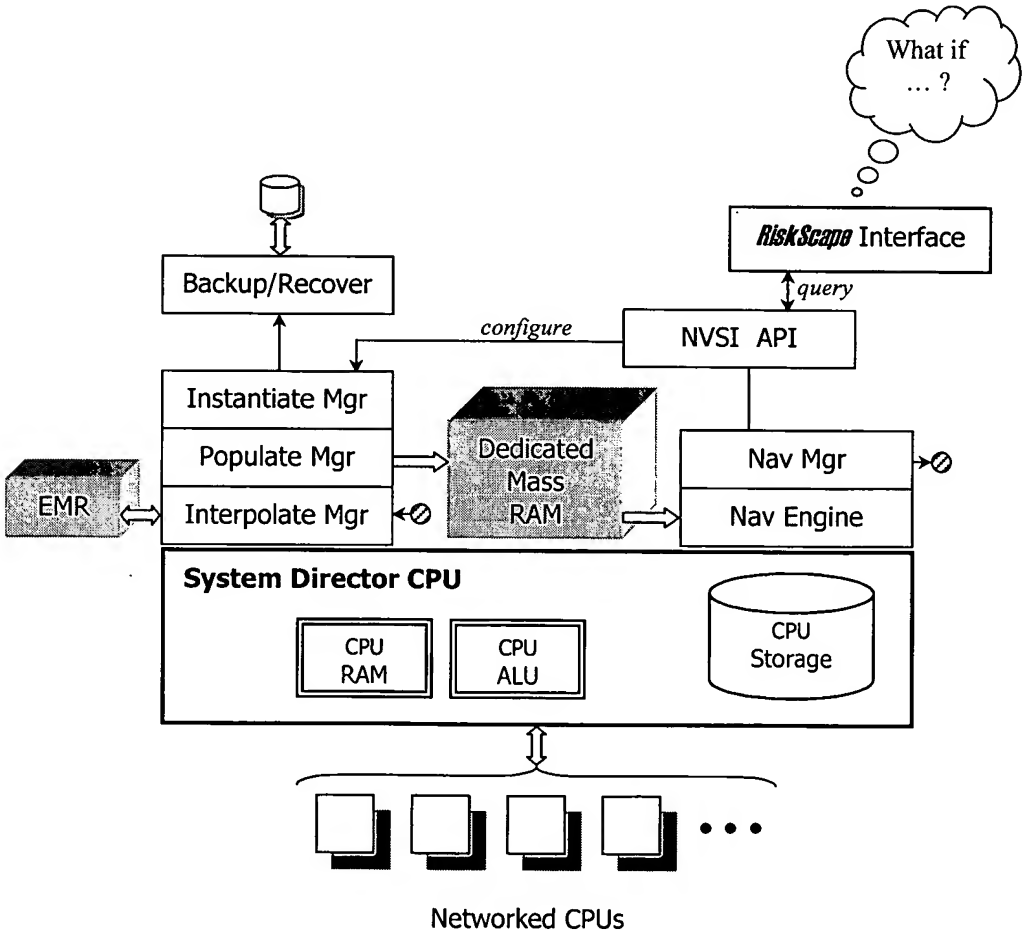

FIGURE 1: Functional block diagram of the NVSI system

The core of the NVSI system is a suite of powerful state-of-the-art applications built around a large (128GB) dedicated (non-swappable) *MassRAM*, that in later versions will be implemented via shared RAM from network resources (or even hard-storage-based virtual memory). The particular structure of the data words and hyperspatial connectivity are implemented from domain-specific parameters set by the client user via the *RiskScape* application. The *Instantiation Manager* then configures the design by creating a set of tables (in CPU RAM) containing metadata, such as data-word definitions, tree structures, pointer parameters, network sharing, and problem-domain specifications & indexes. The state-vectors (hyperspace nodes) are then computed and stored in MassRAM by the *Population Manager*, which calculates all function (model) values for each scenario. The Population Manager works continually, in background, distributing the burden across the shared Network CPUs. On-demand mark & search (via queries from *RiskScape*) of the state-space is then handled by the *Navigation Manager*, in concert with the *Interpolation Manager* (and its associated Extended MassRAM – *EMR* – for finer-grained exploration of selected node-neighborhoods). More detail is presented in subsequent sections.

IV. Data Structures

For each entity in the collection (each instrument-type in a portfolio, for risk applications), the NVSI constructs a path-dependent tree (a rooted, ordered, unidirected graph), also more generally termed a *metastructure* or *scenario-tree*. The branches (edges) represent variations ΔI_x in input-parameter I_x with fanout κ. Every node (vertex) is a state-vector $\boldsymbol{s}$ containing the value (output) of one or more model vectors $\boldsymbol{V}$ for the given instrument (allowing for multiple models to value the same instrument), and the probability $P(s)$ associated with that node (derived from the conditional probability of the particular input parameter variation that led to the current state), for each timestep t_k in a sequence of chosen intervals. An example tree, for fanout $\kappa = 9$, is shown below:

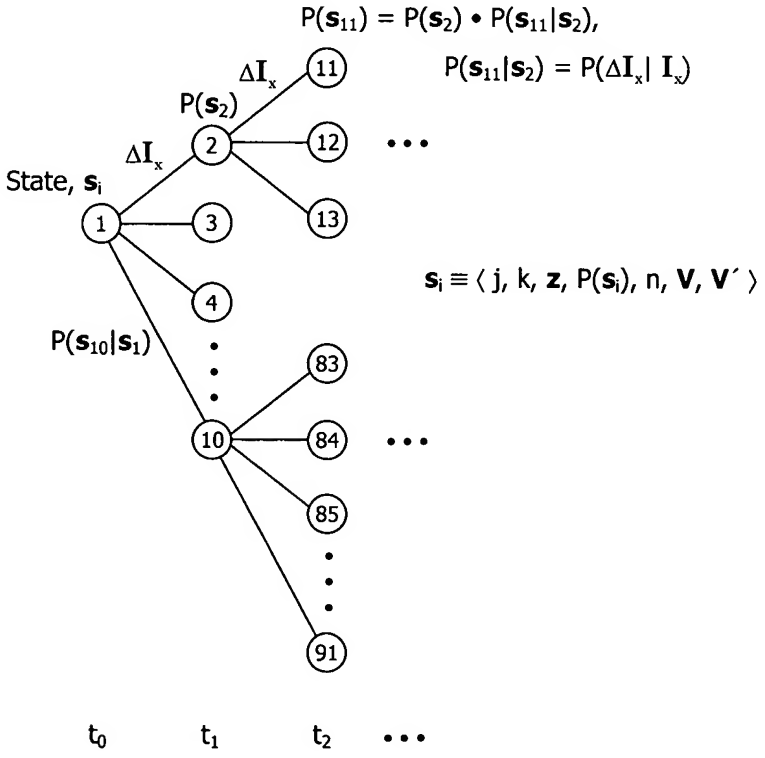

FIGURE 2: An example tree-fragment illustrating how nodes & branches are connected, how states are represented, and how probabilities are derived.

For each node i (a point in the state space, H) there is an associated state s_i, defined by a 7-tuple state-vector:

$$s_i \equiv \langle j, k, z, P(s_i), n, V, V' \rangle,$$

where:

- $1 \le i \le \Omega$, $\Omega \equiv$ cardinality of H = total number of nodes (state-vectors);
- j is the number of the particular branch in a fanout of κ (required to derive the back pointer), $1 \le j \le \kappa$.
- k is the number of the time-step for interval t_k, $1 \le k \le \tau$, $\tau \equiv$ chosen number of time-steps = depth of the tree. Note that the intervals may be non-uniform;
- z is an array of logical (Boolean) flags indicating various computation conditions, such as mode (populate or navigate), update status, interpolation-flag, constraint-satisfaction flag, etc.
- $P(s_i)$ is the probability of state s_i occurring, calculated in the standard way from the joint probability of all ancestor nodes, which is in turn derived from the recursive product of marginal and conditional probabilities:

  $P(s_i) \equiv P(s_i \cap `s_i \cap ``s_i \ldots)$, where the ` operator denotes a parent state (node on the tree),

  $= P(`s_i) \bullet P(s_i \mid `s_i)$, $P(`s_i) = P(``s_i) \bullet P(`s_i \mid ``s_i)$, $0 \le P(s) \le 1$.

  For *RiskScape* applications, the conditional probabilities are just the transition probabilities for changes in financial variables. In general, although the states (output function values) are not usually path-dependent (except for some exotic instruments), the transition probabilities *are* conditional. Thus, the probability of a change in volatility σ, for example, is dependent on the baseline:

  $P(s_i \mid `s_i) = P(\Delta\sigma \mid \sigma)$;

- n is the number of functions or variables included in V, and thus essentially determines the data-word length, where:
- $V \equiv [v_1, v_2, \ldots. v_n]$ is the *value-functional*, a set of one or more v, which are domain-specific function equations, input parameters, output variables, or parameter-pointers, derived from pricing models (or any state-independent equation from the problem domain), where a given v is generally defined by:

  $v \equiv f(t, I(t))$, where:

- $I(t) \equiv [a, b, c, \ldots]$ is an array (vector) of m input variables that are subject to change (ΔI_x) at each time-step, thus generating the various κ branches for each node;
- V' is the dual of V, containing the next-day's "aged" set of values, computed by the *Population Manager* after it computes V (see section VI).

A particular *state* is defined by computing scalar values for all the variables (or functions) v defined in the state-vector, for the particular node in the tree.

Note that κ is just the total number of variations over I. The way that κ is mapped onto I, *i.e.*, how the available granularity in variations is distributed across the input variables, is defined by the client-user.

Three key features of the data structure are:

- Pointers are not stored, but *induced*. That is, successor nodes are indexed by an algorithm (modified from Knuth 68) that computes addresses for child nodes in balanced trees as adjacent words in memory. For parent node i, the child nodes on its subtree are numbered by:

  $$\kappa \bullet (i\text{-}1) + (1+j),\ i > 0, 1 \le j \le \kappa .$$

  This enables fast calculation of pointers, saves an enormous amount of storage, and wastes virtually no memory space. And because the trees are fixed until the state-space is recreated for a new domain and a new set of scenarios, there is very little shuffling or garbage collection required.

- The actual data values are stored as bit-string integers, even for floating-point numbers, which are stored as fixed-point integer pairs (mantissa & exponent) of greatly shortened length to handle just as much accuracy as needed (typically 10 bits – for 1 part in 1000 – instead of the 32 that is standard in desktop ALUs). These short integers are passed to the system ALU for simple integer arithmetic, thus dramatically decreasing the computational demand.

- Exponentials, logarithms and roots are retrieved via stored-table lookup, again at just the accuracy required.

The total number of nodes (or states), Ω, in the hyperspace H is constrained by the total amount (in bytes) of dedicated MassRAM (denoted by *Mem*) available, and Λ, the characteristic length (in bytes) of the data words. Thus,

$$\Omega(H) = \text{total number of nodes (states) in the hyperspace} \equiv Mem / \Lambda .$$

As shown later, a typical instantiation for *RiskScape* yields $\Omega \sim 10^9$ states.

A *scenario* is the path of events, or conditional changes, that leads to a given state. Thus, each node represents the result of one scenario. The entire set of scenarios yields a scenario-tree, which is exactly one metastructure. Note that, typically, different metastructures are defined on the same landscape of events, and thus share the same set of scenarios. The total number of separate trees is therefore constrained by both $\Omega(H)$ and the chosen fanout κ (derived from the desired granularity in input variation) and τ (the total number of time-steps desired in the analysis).

For balanced trees, the number of nodes M in a tree is given by the sum of a geometric progression of base κ:

$$M = [\kappa^{(\tau+1)} - 1]/(\kappa\text{-}1) = \text{total number of scenarios} \cong \kappa^{\tau},\ \text{for } \kappa \gg 1 .$$

The total number of available scenario-trees (metastructures), N, is therefore:

$$N = \Omega / M .$$

V. *RiskScape* Structures & Design Factors

The NVSI, along with the *RiskScape* Intelligent Financial Risk Application, was originally designed to optimize the computation of future portfolio values over a range of scenarios designed to uncover "dragons", the potential catastrophes that can result from less likely events (changes in relevant financial variables) that lie in the "fat tail" of quasi-lognormal (kurtotic) distributions presumed to underlie most financial variations. Typically, large global portfolios are composed of 100,000 to 10M instruments (securities), and include a wide range of types: stocks (equities), bonds, futures, currencies, swaps, convertible debt, interest-rate instruments, options (on all of these), and other exotics. Various models of these instruments exist, and the output variables include price and the various moments (the "greeks") of the model function. Typical input variables, which are changed systematically in a simulation to construct a scenario tree, include p_u (price of the underlying asset), the asset volatility σ, interest rate i_r, and interest-rate volatility σ_{i_r}. Some models have-closed-form solutions that are relatively straight-forward to evaluate (such as Black-Scholes). Others, such as those for interest-rate options, require some form of stochastic simulation (such as Monte Carlo [Berkowitz 98]), finite-difference methods, or trinomial tree expansions for determining term structure (such as Black, Derman, Toy), and are thus far more cumbersome to compute.

Probability values

In the prototype version designed for use in financial risk analysis, the data structure has elements that are key for future-risk computation. For portfolio evaluation, the conditional probabilities $P(\Delta I_x | I_x)$ correspond to the *actuarial* transition-probabilities derived from tables of historically-calibrated movements in financial variables, or extracted from relevant time-series of changes in those variables. For risk analysis, the *risk-adjusted* probabilities are also important, so the *RiskScape* interface allows the user to choose either, or both. If both are specified, one of the alternative probabilities is stored in one of the n variables v.

With a κ (fanout) of only 12-16, and four input variables, for example, there is not enough granularity to allow for joint transitions (such as $\Delta\sigma = 4$ & $\Delta p_u = -0.5$). Thus, in the prototype, branches typically denote orthogonal moves, that is, changes in one input variable at a time. However, implicit nonorthogonality can be approximated by using historically-derived correlations between financial variables. This approach relies on the fact that, with large moves, correlations between financial variables become tighter. Alternatively, κ can be increased to allow for joint transitions, consonant with available resources.

To make the computation tractable for large portfolios (1M+ instruments), and congruent with the theme of future-risk analysis, which is to look at possible catastrophes ("dragons") arising from large moves in the input variables, only changes of $\Delta I_x > 3\sigma$ are typically evaluated. Although available tables of transition probabilities do not always contain such data, the required probabilities can be easily extrapolated, as the distributions (or at least the first moments) are known.

Instrument Proxies

Even with 128GB RAM and modest tree size ($\kappa = 12$ and $\tau = 6$), for large heterogeneous portfolios an unaugmented NVSI can support no more than ~ 100-1000 metastructures (separate scenario trees), depending on the word length Λ (see section VII). To compensate, one of the innovations in the architecture is therefore to use "virtual" or *proxy* instruments: state-vectors with $\boldsymbol{V}$ composed of generic model functions for all the securities based upon the same underlying asset (or base index). The actual instrument prices can then be calculated during navigation, by evaluating the corresponding function and converting back using standard scaling and correlation coefficients. When a proxy state is computed (during the hyperspace-population phase), the various prices and moments of a given model are calculated over a range of values, thus creating a parametric space that brackets the range of possible magnitudes for the portfolio instruments. During the navigation phase, the actual instrument prices (and other moments) are then calculated via extraction of values from the pre-computed parametric-space, using the state-dependent input parameters stored in the proxy state-vector.

For example, suppose the actual instrument is a standard European option, defined by free parameters K (= strike-price), S (=p_u), r (= i_r), δ (= *div yield*), T (= time-to-expiration), and σ (= volatility). If the model of choice is Black-Scholes, the option price can be obtained from a "normalized" space defined by only three parameters: $\{K/S, (r-\delta)T, \sigma\sqrt{T}\}$ (see *Addendum A1*). During the population phase, the parametric pricing space is pre-computed (creating a "parametric-hypercube") and mapped to the proxy, and the values of the input vector $\boldsymbol{I}$ are stored in $\boldsymbol{s}$ for each node in the scenario tree. Then, during navigation, the price p of the actual portfolio option instrument is calculated (marked to scenario) simply by extracting a virtual price p_{proxy} from the associated parametric-hypercube using the input values (K, S, r, δ, T, σ) and then transforming the result using known scaling (α) and correlation (β) factors. That is,

$$p = p_{proxy} \bullet \alpha \bullet \beta$$

Recall that each different proxy represents instruments with different underlying assets, and each proxy is evaluated over one scenario-tree. Thus, the number N of available metastructures determines the number of available proxies onto which the portfolio can be mapped.

There can be many different classes, or types, of instruments in a portfolio [Hull 93], including:

1. Equities & their derivatives (***d***)
2. Debt Instruments & ***d***
3. Swaps & ***d***
4. Currencies & ***d***
5. Collateralized Mortgage-Backed Obligations & ***d***
6. Exotics (such as Interest Rate, Barrier, Lookback and Knockout options).

Some types are much more cumbersome to compute, as indicated above. For example, some of the exotics require complex use of input parameters, such as interest-rate derivatives that use a path-dependent interest-rate curve (or actually, an interest-rate vector). Models such as Heath-Jarrow-Morton and Black-Derman-Toy are valued with state-dependent trees, which the NVSI architecture already supports.

The proxy state-vector must contain models or parametric references for as many types of instruments as there are in the portfolio. For example, a proxy might contain variables for cash, futures, standard

options (all Type 1), sovereign debt (Type 2), and interest-rate options (Type 6). All instruments of Type 1, 2, & 6 in the portfolio that are also based upon the same underlying asset or base index (such as the *S&P 500*), are linked to this proxy. Thus, there need to be as many proxies as there are underlying indexes in the portfolio (typically in the hundreds).

Data Word Structure

The structure and size of a data word (the bit-wise representation of a state-vector) is specified by the Data-word Definition Table (see ***MetaTables***), which is constructed by the *Instantiation Manager* according to specifications received from the client-user. For each proxy, the *Population* and *Navigation Managers* carry this table around, so to speak, as a part of their operation.

Data-word structure is flexible. The default is as follows:

The back-pointer, j, indexes the branch (of κ) from the parent node that a state occupies. The prototype allows for a maximum fanout of $\kappa = 64$, so the length of j in bits is:

len j = 6 bits.

The prototype allows for up to $\tau = 16$ time steps, so

len k = 4 bits.

The flag-vector, z, is one byte (to handle up to 8 conditions):

len z = 8 bits.

Probabilities must reflect likelihoods derived from large-move events, and need be no more accurate than 1 part in 1000 (three significant digits). Thus,

len $P(\mathbf{s})$ = 10 bits.

The number n of variables in the value-vector $\boldsymbol{V}$ is unbounded, but the prototype allows for 256. Thus,

len n = 8 bits.

Strictly, n is redundant, because the data-word-definition table (created by the *Instantiation Manager*) specifies the number of variables in the state-vector. Yet, placing n in the state-vector promotes data integrity, and its storage penalty is small.

The first part of a state-vector (everything but $\boldsymbol{V}$), denoted by $\langle \boldsymbol{s}$, thus has a length:

len $\langle \boldsymbol{s}$ = 36 bits.

Finally, the variables in $\boldsymbol{V}$ can represent function outputs, parameter ratios, input variables, statistical-distribution points, partial-derivative values, or parameter-reference pointers. In general, the magnitude ranges are known and specified in the data-word-definition. Thus, the only part stored in the state-

vector is the integer mantissa, again typically to three significant digits for dragon-hunting. In those cases where the range is arbitrary, or infinite, an integer exponent to accommodate $10^{\pm 15}$ (5 bits) is also stored. The default, then, is:

len v = 10 bits *or* 15 bits .

For the complex proxy described above, which we shall use as a conservative benchmark, the state-vector would contain one function variable for a cash index, one variable for each of several (say four) futures, four variables from the input vector $\boldsymbol{I} = [p_u, \sigma, i_r, \sigma_{i_r}]$ for the Type 1 options, perhaps an additional four (different) input variables for the Type 2 options, and an interest-rate vector of, say, eight variables, that represents the path-dependent interest-rate curve for the Type 6 options. We can also imagine that some of the exotics have models so complex that the parametric-hypercube yields a family of curves; or perhaps an instrument-type is so new that its model has not yet been parameterized. In either case, variables (say ten) representing curve-selection values, or five slope-intercept pairs for linearized segments of the curves, need to be stored.

There is also one variable required to store the alternate probability measure.

Then,

$n \equiv$ number of variables $\cong 1 + 4 + 4 + 4 + 8 + 10$ (*or* 5•2) $+ 1 = 32$.

If most of the price-related variables (cash, futures, curve-segments) require stored exponents, then

len $\boldsymbol{V} \cong (15 \bullet 15) + (17 \bullet 10) = 395$ bits.

Accounting for the length of $\boldsymbol{V}'$ (the time-dual of $\boldsymbol{V}$), this yields

len $\boldsymbol{s}$ = len $\langle \boldsymbol{s} + 2 \bullet$len $\boldsymbol{V} = 36 + 2 \bullet 395 = 826$ bits.

The actual word-length, in bytes, for our benchmark proxy state-vector, is then

$\Lambda = \lceil$ len $\boldsymbol{s} / 8 \rceil = 104$ bytes.

This will be used to calculate relevant performance parameters in section VII.

MetaTables

The user specifies the nature of the problem domain, and all relevant data, via the *RiskScape* Interface, using a Scenario Description Language (*SDL*). The *Instantiation Manager* then creates and configures a set of tables to implement the specifications. Typical tables include:

- Data-word Definition (including field structure & bit-masks for storage and field extraction)
- Hyperspace Definition (topology, κ, τ)
- Node Index (as an adjunct to pointer induction)
- Proxy Definition (types, model variables, parameters)
- Scenario Definition (input vectors $\boldsymbol{I}$, the mapping of $\boldsymbol{I}$ to κ)

- Input Variable Data (updated & adjusted values for all possible I_x, such as σ, i_r, *etc.*)
- Probability Transition Data (derived from commercial tables or extracted from time-series)
- Arithmetic Lookup (exponentials, logarithms, roots)
- Conversion Factors (scaling, coefficient, parameter-estimation data)
- Portfolio Data (coded description of all instruments and their specifications, including proxy links, and whether or not the instrument is to be included in the reference sample).
- Pricing-Model Parametric Data (pre-computed hypercubes of pricing-model values and moments)

Some of the tables are quite large, such as the Portfolio Data (1 million records or more) and the Node Index (a billion records). All are stored on hard disk, but some are also resident in CPU RAM (not MassRAM, which holds only the hyperspace *H*), such as the Data-word Definition, Proxy Definition, Probability Transition, Input Variable and Scenario Definition tables.

VI. The Infrastructural-Technology Suite

The core NVSI modules are:

- *RiskScape* Application: the user specifies (using SDL) the nature of the problem domain, the type of structures involved, the fanout κ, the time-depth τ, the mapping of κ to the input-vector I, the portfolio to evaluate, the scenarios desired, the event and probability thresholds, and so forth.

- *Instantiation Manager*: configures all the definition tables and MassRAM to handle the domain specifications from *RiskScape*.

- *Population Manager*: computes all state-vectors in background (off-line) processing, according to the rules and tables specified by the *Instantiation Manager*. One of the key optimizing techniques used is to simulate interleaved memory by having the *Population Manage* first compute all the values of $\boldsymbol{V}$, to have them available to the *Navigation Manager* (see below) as soon as possible, and then it computes the next updated (dual) set of values $\boldsymbol{V}'$ – the same variables, same value of t – but updated for the next day's data, as the portfolio is "aged". The *Navigation Manager* then selects the second set of values (if the ready-flag has been set by the *Population Manager*), so that portfolio update is performed synchronously, over the entire hyperspace.

- *Navigation Manager*: invokes queries from the client-user (in this example via the *RiskScape Application*) to search the hyperspace *H*, evaluate H over each scenario, mark the nodes that either meet the event & probability thresholds, or that warrant further exploration (via the *Interpolation Manager*). It is the *Navigation Manager* that responds in near-real-time to user requests, and "walks the landscape" of the hyperspace to hunt for dragons.

- *Interpolation Manager*: in concert with the *Navigation* and *Population Managers*, it applies *mesh enhancement* to selected nodes. That is, it creates an expanded tree-fragment with a finer granularity in the neighborhood of the node, to yield more accurate values that may lie "between" certain states. This enhancement may entail expanding the *spatial* granularity, via a larger κ with finer gradations in $\Delta \boldsymbol{I}$, or using correlations to simulate joint transitions.

Enhancement may also be *temporal*, expanding τ locally by creating a tree-fragment with finer-grained time-steps (and this may also use correlations to interpolate variations in input variables).

VII. Performance Parameters & Evaluation

In this section, boundary conditions and typical mid-range values for spatial (storage) requirements and temporal performance are derived.

Spatial parameters

Recall that:

$\Lambda \equiv$ characteristic data-word length, and

$\Omega(H)$ = number of available states in $H \equiv Mem / \Lambda$, Mem = available memory (in bytes) .

Thus, for the prototype MassRAM of size 128GB (= 137×10^9 bytes), and a typical Λ (for our benchmark proxy) of 104 bytes:

$\Omega = (137 \times 10^9 / 104) \cong 10^9 =$ **1 billion nodes (or states) available in the hyperspace.**

Note that with a $\Lambda \sim 50$ bytes (for very simple Type 1-only proxies) to 200 bytes (for very complex proxies representing instrument Types 2, 5, & 6), the result is still about the same.

For the same size scenario tree as used earlier, with $\kappa = 12$ (allowing three gradations, or moves, for each of four input variable) and $\tau = 6$ (allowing for six time-steps, such as 1 day, 3 days, 1 week, $t_{VaR} \sim$ 20 days, 3 months, 6 months), then

M = tree size $\cong 12^6 \cong 3 \times 10^6 =$ **3 million nodes in the scenario tree.**

Thus,

N = number of scenario-trees = $\Omega / M = 10^9 / (3 \times 10^6) \cong 300$ = number of proxies allowed.

The number of proxies available to map the portfolio onto is therefore about 300, for $\kappa = 12$ and $\tau = 6$. Can a large global portfolio be mapped to only 300 proxies, that is, only 300 base indexes? Absolutely. There are only six types of proxies, and linking these to 300 underlying assets/indexes would cover most of the developed world.

To bound this result, consider a scenario tree with a larger κ of 16:

$M = 16^6 \cong 1.7 \times 10^7 \Rightarrow N \sim 60$, still large enough to represent a fairly diverse portfolio.

Suppose that we don't need six time steps, but only four (1 week, t_{VaR}, 1 month, 3 months):

$M = 16^4 \cong 65{,}536 \Rightarrow N \sim 15{,}000.$

Thus, reducing the number of time-steps significantly increases the number of proxies that can be created. If we wish to increase the granularity of input variations, perhaps allowing for joint transitions (like $\Delta\sigma$ & Δp_u), then κ can be increased to 32 or 64. If a client-user simply wanted to value a portfolio at one timestep ($\tau = 1$) t_{VaR}, and with very high precision in the scenario mesh ($\kappa = 64$), then

$N = 10^9 / 64 \cong 16$ million .

At this level, a sizable portfolio can be evaluated for future risk directly, without the need for proxies.

The realistic **spectrum for N** is thus:

$$1 = N_{32,6} < (30 = N_{32,5}) < (N_{16,6} \sim 60) \leq (N_{\kappa,\tau} = N_{12,6} \sim 300) \leq (N_{16,4} \sim 15{,}000) \leq (N_{64,1} = 16 \text{ million}).$$

Since many portfolios require less than 100 base indexes, then with a typical N of ~ 300 available for our benchmark tree of $\kappa = 12$ and $\tau = 6$, the effective fanout κ can be increased (thus increasing the input-variation granularity), by indexing an array of proxies to one instrument-class, each identical except that the scenario-trees use different ΔI increments. For example, one tree could assign each branch to variations of $(0, \pm 3\sigma)$, another tree have branches for $\pm 8\sigma$, and another for $\pm 12\sigma$.

As we show in the temporal performance calculations, the virtual effective throughput for navigation is high enough that it will be possible to implement MassRAM not only as a shared network resource, but eventually, as virtual memory using hard sequential storage. That is, because the hyperspace trees are fixed and space-filling for a given problem domain, the RAM transaction volume is low, with little random access. Thus, by using *JINI*™ technology (for example) over shared network resources, NVSI v2.0 can use FIFO paging from optimized disk storage, performing a look-ahead page-fetch in 16GB (or larger) segments, while still not slowing the Navigation Manager. Under such an operating system, *Mem* is virtually unbounded, and we could realistically process 10^{12} nodes (which still only requires a 40-bit address) or more.

Temporal parameters

Performance of the NVSI is partitioned into the two primary phases: the time T_{pop} to populate (compute and fill all the state-vectors in) the state-space , and the time T_{nav} to navigate the space (evaluate the domain collection – such as a financial portfolio – at each scenario, and flag selected nodes for states that meet the criteria).

The key to the NVSI idea is that the apparent (effective) throughput in response to a user query is driven by the navigation-time, as the population-time reflects background (off-line) computing.

As a reference, performance parameters are derived for the *RiskScape* problem domain, using the benchmark proxy already described.

POPULATE

The population time has four principle components:

1. A one-time setup by the *Instantiation Manager*;
2. A one-time pre-computation of the parametric-hypercubes for all relevant pricing models;
3. A recurring (for each daily update, as the portfolio is aged) recalculation of all values in V for every node in H; and
4. Background I/O and network-sharing overhead, which varies according to the amount of real-world data capture, and the size of H.

It is the third component that is most characteristic of T_{pop}. For our benchmark proxy, the calculation requirements for each of the $\langle s$ values is:

j: 3 integer operations (INT)
k: 1 INT
z: 8 INT

$P(s)$: 1 lookup (~ 1 INT) & one floating-point multiply (~ 1 FLOP, FLoating-point OPeration)
n: 1 lookup & store ~ 1 INT

For V, the calculation times are highly varied. The cash variable requires 1 INT, the futures each require ~ 3 FLOP + 1 lookup (~ 1 INT), and the alternate probability ~ 1 INT + 1 FLOP. Each of the input parameters involves 1 FLOP to calculate, and similarly for the interest-rate values. The curve-selectors and/or slope-intercept pairs require ~ 2 FLOP. Each of the 32 variables values takes 1 INT to store in the state-vector. Thus, the total (recurring) time to populate one state-vector for our benchmark is:

1(1 INT) + 4(3 FLOP + 1 INT) + 4(1 FLOP) + 4(1 FLOP) + 8(1 FLOP) +
10(2 FLOP) + 1(1 INT + 1 FLOP) + 32 (1 INT)

= 38 INT (~ 4 FLOP) + 49 FLOP ≅ 53 FLOP.

The total FLOPs, $F_{pop}(3)$, to populate all nodes is then:

$F_{pop}(3) = \Omega * 53$ FLOP $= 53 \times 10^9$ FLOP = 53GFLOP.

We take as a baseline-reference CPU a typical mid-range, stand-alone workstation, with a computational throughput of $R = 10^7$ FLOPS (FLoating-point Operations Per Second) = 10 MFLOPS. Then,

$T_{pop}(3) = (53 \times 10^9$ FLOP$) / (10^7$ FLOP / sec$) = 5300$ seconds ~ 1.5 hours.

To calculate the pre-computation time for the various pricing-model parametric-hypercubes, we require the characteristic computation time for typical models, and a choice of granularity in the parameter space. For Black-Scholes options, computing one price and the associated moments (Δ, Γ, *vega*, Θ, ρ, τ) takes ~ 70 FLOP; one barrier or lookback option & moments: ~ 1000 FLOP, and one exotic derivative (such as Black-Derman-Toy or Heath-Jarrow-Morton): ~ 2,000 – 20,000 FLOP [Ferrentino 99].

One of each type is needed for the benchmark proxy. Assuming three parameters for each (actually, six is more appropriate for exotic derivatives, but the hypercube becomes enormous, so each point is made a vector that embeds a family of curves), and assuming a granularity of 100 increments per parameter (dimension), then

the number of points for each parametric-hypercube = 10^6 (1 million).

The total computation to create a Type 1 hypercube is thus 70×10^6 FLOP, and for a Type 2 hypercube, 1000×10^6 FLOP, and a Type 6, 20000×10^6 FLOP .

Thus, the amount $F_{pop}(2)$ of parametric-hypercube precalculation is dominated by the exotics,

$F_{pop}(2)$ = FLOPs (Type 6) + FLOPs (Type 2) $\sim 2.1 \times 10^{10}$ FLOP = 21GFLOP

For the 10MFLOP CPU, this yields

$T_{pop}(2)$ = 2100 seconds ~ ½ hour.

If we estimate the Instantiation time $T_{pop}(1)$ at about the same (to fill all the tables, some with millions of entries), ~ ½ hour, and that system overhead (category 4) doubles all the other times, then we have:

$T_{pop}(\text{once}) = 2 \bullet [\, T_{pop}(1) + T_{pop}(2) + T_{pop}(3)] \cong 5$ hours, and

$T_{pop}(\text{recurring}) = 2 \bullet T_{pop}(3) = 3$ hours.

That is, the entire hyperspace H of states can be repopulated daily, allowing for real-time aging of the portfolio.

NAVIGATE

Navigation of H is the process of evaluating (pricing fully) the entire portfolio over the entire scenario-tree, and flagging the nodes (states) that satisfy the client-user query (on both probabilities and values). The Navigation time, T_{nav}, is thus dominated by the time required to price the portfolio, which entails pricing each instrument, and summing over the entire collection. Pricing an instrument involves calculating the virtual price (and all moments) from the proxy, and transforming with known multiplier coefficients (scaling, α and correlation, β). Doing this in turn requires taking each value stored in V (usually a set of input variables I), combining it with the free parameters of the instrument (such as X & T for a Black-Scholes option), and calculating the relevant selection parameters, used to either access a parametric-hypercube, or for newer instruments or open-form models, to calculate the final function output directly.

The process of evaluating the portfolio price for one scenario, that is, for one node (state) in the scenario-tree, we term *mark-to-scenario* (m-t-s). Evaluation of the portfolio over all scenarios, we term *mark-to-landscape* (m-t-l).

Flagging a node involves only two compares and a bit-flip in z, so $T_{nav} \cong T_{m\text{-}t\text{-}l}$, which is given by:

$$T_{m\text{-}t\text{-}l} = [T_{price} \bullet \omega + T_{sum}] \bullet M,$$

where: T_{price} is the characteristic time to price one actual instrument (and all of its moments),
ω is the total number of instruments in the portfolio, and
T_{sum} is the time to add all of the instrument prices to yield a total value for one state.

Note that only one instrument-type will be calculated for any given proxy-vector (although the proxy may contain data to price all the various types it represents).

One of the essential aspects of the NVSI architecture is that, because the function-spaces are pre-computed (during the populate phase), the time to extract the price-values is nearly independent of model complexity. Instead, evaluation time is dependent on the number of parameters required to calculate for extraction of the proxy price.

Of course, the cash and futures values are stored in s directly, so pricing their corresponding instruments is trivial. Thus, a more realistic estimate is obtained by calculating the T_{price} for the options, all of which are comparable to each other.

For the Type 1 option, the three parameters for extracting the price (and any other moment) are: $\{K/S, (r-\delta)T, \sigma\sqrt{T}\}$. Each parameter requires about 1–2 FLOP to calculate, for a triplet total of ~ 4 FLOP. All of the moments are obtained with the same parameters at the same time from the parametric hypercube, so the time to obtain the entire proxy-price is just 4 FLOP. To value the instrument-price, each proxy-price moment is then multiplied by a combined factor $\gamma = \alpha \bullet \beta$, which adds 1 FLOP to the process.

Thus, to price all eight moments (p, and 7 greeks, for advanced models) of an actual option instrument, the amount of computation required (in FLOPs), F_{price}, is given by:

$$F_{price} = (4 + 8) \text{ FLOP} = 12 \text{ FLOP}.$$

For a 1M ($\omega = 10^6$) instrument portfolio, the FLOPs to mark the entire portfolio to all scenarios for our benchmark is then:

$$F_{m\text{-}t\text{-}l} = [F_{price} \bullet \omega + F_{sum}] \bullet M$$

$$= [12 \text{ FLOP} \bullet 10^6 + 10^6] \bullet (3 \times 10^6) = 39 \times 10^{12} \text{ FLOP} = 39 \text{ TFLOP}.$$

The computation required to fully risk-evaluate a 1 million instrument portfolio over an entire scenario tree ($\kappa = 12$, $\tau = 6$) is: **39 TFLOP**.

On the reference CPU, the time required is then:

$$T_{m\text{-}t\text{-}l} = F_{m\text{-}t\text{-}l} / R = (39 \times 10^{12} \text{ FLOP}) / (10^7 \text{ FLOP / sec}) = 39 \times 10^5 \text{ seconds} \cong 1100 \text{ hours}.$$

At this point, two key components of the NVSI come into play:

- A standard procedure for large-scale computing is *two-pass simulation.* First, a statistically-representative sample of the full problem domain is evaluated, marking (flagging) the nodes of interest, and then the full domain is evaluated over the flagged nodes. This is equivalent to a *mesh-enhancement* for the navigation phase. In portfolio-risk analysis, a 1000-fold reduction of the portfolio to a sample is well within norms. Under this compression, then,

  $T_{\text{m-t-l}}$ (sample) = $(39 \times 10^9) / 10^7 \cong 1.1$ hour.

  Assume that one in a thousand nodes (states) are flagged for full evaluation of the entire portfolio (because the sample portfolio value meets the client-user query criteria). Then,

  $T_{\text{m-t-l}}$ (entire) ≅ 1.1 hour, again.

  Thus, the total *navigation* time is:

  $T_{\text{nav}} \cong T_{\text{m-t-l}} = T_{\text{m-t-l}}$ (sample) + $T_{\text{m-t-l}}$ (entire) = 2.2 hours, for 1 CPU.

- *The netcentric component:* With a typical 100BaseT LAN of 100 workstations, with a 50% availability (sharing-efficiency), we have:

  T_{nav} (Netcentric) = (2.2 hours) / (100 • 0.5) ≅ 158 seconds ~ 2.6 minutes.

Using netcentric computing and problem-domain optimization, the entire portfolio can be navigated in about 2½ minutes, that is, in near-realtime.

Recall that the client-user measures performance in terms of the response to query, that is, on-demand access to H over all scenarios. Thus, we calculate an effective (virtual) throughput for this computation, optimized for portfolio risk-analysis with 100 shared reference-CPUs (in terms of virtual-FLOPS, or VFLOPS) as:

R_{NVSI} (Risk Domain / 100) = $F_{\text{m-t-l}} / T_{\text{nav}}$ = 39 TFLOP / 158 sec ≅ **250G VFLOPS**.

A bounded spectrum for performance (in units of GVFLOPS) is then:

5 = R_{NVSI} (RD×1) < **25** = R_{NVSI} (RD×10) < **250** = R_{NVSI} (RD×100)

Note that the value for 1 CPU assumes a dedicated machine.

Comparison with Commercial Solutions

For comparison, if one were to simply "scale-up" existing instrument-valuation software, and run it on the best mainframe (dedicated hardware supercomputer) available, then a similar calculation to the above would involve very different computation times for different instruments. If we assume a typical global portfolio with 1M instruments, and a mix of 50% Type 1, 40% Type 2 and 10% Type 6, then the total computational demand is given by:

$$(0.5 \bullet 1000\text{FLOP} + 0.4 \bullet 70\text{FLOP} + 0.1 \bullet 20{,}000\text{FLOP}) \bullet 10^6 + 10^6) \bullet (3 \times 10^6)$$

$$= 7587 \times 10^{12}\ \text{FLOP} = 7587\ \text{TFLOP}\ .$$

A *Cray T3E* Supercomputer has a peak (burst) throughput of ~ 2.2 TFLOPS, and a sustainable rate (which we estimate is appropriate for the I/O requirements of this problem space) of $R \sim 250$GFLOPS. Therefore, even on a *T3E* the Risk Domain problem would require:

$$T_{nav}\ (\text{Cray T3E}) = (7587 \times 10^{12}\ \text{FLOP}) / (250 \times 10^9\ \text{FLOP / sec}) \cong 30{,}000\ \text{sec} \cong 8.4\ \text{hours.}$$

On a more affordable (and obtainable for a financial institution) *Cray T90* (for which we estimate in this problem space a sustainable $R \sim 18$ GFLOPS), the problem would take nearly 5 days, and on a conventional high-end business mainframe, about 100 days.

PRICE / PERFORMANCE

The most straightforward measure of price/performance is simply cost/*R*. The projected cost of version 1.0 of NVSI is ~ $10M. The cost of a fully-populated *Cray T90* is ~ $20M, and a fully-populated *Cray T3E* ~ $100M. Thus:

$$\text{P/P NVSI} = (\$10 \times 10^6)/(250 \times 10^9\ \text{VFLOPS}) = 0.00004\ \$/\text{FLOP} = 0.004\ ¢\ /\ \text{FLOP}$$

$$\text{P/P C90} = (\$20 \times 10^6)/(18 \times 10^9\ \text{FLOPS}) = 0.00111\ \$/\text{FLOP}$$

$$\text{P/P T3E} = (\$100 \times 10^6)/(250 \times 10^9\ \text{FLOPS}) = 0.00040\ \$/\text{FLOP}$$

Therefore, the NVSI is about 10 times more cost-effective than the only other machine that can solve the problem in reasonable time. And this does not even include the extended costs for the *Cray* machine of software development (including staff), machine-room support, & depreciation.

In contrast, all of our NVSI calculations have been conservative: in MassRAM use, problem-complexity, structural-requirements, estimated calculation times, and network sharing & resources.

With these factors considered, we project that the cost of an NVSI implementation will be 50-100 times less than any comparable hardware-based commercial solution.

VIII. Summary

What have we really achieved? In theoretical terms the three fundamental breakthroughs are:

1. The NVSI architecture allows flexible (and extensible) reconfiguration of the system memory in such a way that the actual topology of the computational surface is modifiable. This means that a true virtual machine is constructed, one which is unique for each problem domain. One of the principle assertions of computability is that the more closely the architecture of the machine matches the architecture of the problem being solved, the more efficiently the machine will solve that problem. In our case we accomplish this because the core of the NVSI is a problem-domain-optimized solution manifold. This solution manifold has been created, or "instantiated", in system memory (hence the name for one of the component modules: the Instantiation Manager). In the case of the *RiskScape* application, this solution manifold is a combination of a rooted, ordered, unidirected graph and a set of pricing-model hypercubes. This manifold is unique to the problem of financial portfolio stress-testing, and is one of the principal reasons that multi-gigaflop throughput is attained.

2. The power of netcentric distributed computation is brought to bear in two ways. First, we break the metaproblem into two independent components: that which can be pre-computed and stored in memory, and the real-time query process. The Population Manager takes responsibility for the former, and the Navigation Manager for the latter. This enables a temporal compression to be applied to the problem. In the case of *RiskScape* the solution manifold requires on the order of 75 GigaFLOP (75×10^9 floating-point-operations) to fully populate all of the pricing vectors in state space. However, to query (*i.e.* fully Navigate) this manifold such that a one million instrument portfolio is evaluated against three million scenarios, takes less than an hour of user time. This theoretical performance assumes that only 10 "Pentium-Class" CPUs are dedicated to the Navigation process. With 100 Pentium IIIs the process runs in near real-time.

3. This is all accomplished with off-the-shelf hardware. In the first version of the NVSI platform, the system will require one dedicated machine as the memory manager. However, we anticipate that with coming advances in Network Operating Systems[2] this constraint will be removed. At that point, perhaps 18 to 24 months from the date of this writing, NVSI will be truly and completely a virtual supercomputer. Then, as the software continues to evolve, so does the "machine".

[2] We especially anticipate advances in Sun Microsystems *JINI*™, which enables peer-to-peer communication between component hardware. Sun Microsystems engineers suggest that this peer-to-peer addressing will evolve to the point that the RAM on any given physical machine can be directly addressed by other physical machines. This will mean that our NVSI platform will be fully "virtualized", requiring no dedicated hardware at all.

IX. References

Aho, V. A., J. E. Hopcroft, & J. D. Ullman (1974). *The Design and Analysis of Computer Algorithms.* Reading, MA: Addison-Wesley.

Berkowitz, G. C. (1998). Monte Carlo simulation for computing value-at-risk: a tutorial review. *NSL TR 98-1*, Denver.

Berkowitz, G. C. & Wurtz, C. C (1998). The impact of near-term technology development on global risk assessment. (*privately distributed*)

Ferrentino, G. (1999). (Personal communications on parametric optimization for various pricing models.)

Hillis, W. D. (1985). *The Connection Machine.* Cambridge: MIT Press.

Hull, J. C. (1993). *Options, Futures, and Other Derivative Securities (2nd Ed).* Englewood Cliffs: Prentice-Hall.

Jorion, P. (1997). *Value at Risk.* New York: McGraw-Hill.

Knuth, D. E. (1968). *The Art of Computer Programming (3 vols).* Reading, MA: Addison-Wesley.

Lawrence, D. (1996). *Measuring and Managing Derivative Market Risk.* London: ITP.

Wurtz, C. C. (1998). Laws, myths & nightmares: the coming paradigm shift in risk management. (*privately distributed*)